Mexico Autonomous Institute of Technology
Design and Engineering of Intelligent Information Systems
COM-35702

Homework 2
Logical Architecture and UIMA Analysis Engines Design & Implementation
Report

Fernando Aguilar – 114297

1. Introduction

In the previous homework, a type system was designed in order to capture all the annotations required in a Question Answering system. In this homework, a set of annotators were created for searching and generating the annotations on a set of documents whose content consist of a question and a set of answers, indicating whether or not are correct. Those annotators were implemented as Java classes, with its corresponding analysis engine descriptor. Also, UIMA Java SDK (uimaj) was used to test and run the annotators.

2. Annotators description

It's important to note that annotators implements the logic required to create the processing pipeline described as follows:

1. **Test Elements**: The system will recognize lines of text that represent either a question or an answer. Each line forms a sentence.
2. **Tokens**: For each sentence, a set of tokens will be extracted. A token is a substring of a sentence, which are formed by breaking the sentence on whitespaces and punctuation signs.
3. **N-grams**: Sets of 1-grams, 2-grams and 3-grams will be formed with the tokens created in the previous step.
4. **Answer Scoring**: For each answer, a score will be calculated using the annotations generated in the previous steps. Note that each answer belongs to only one question.
5. **Evaluation**: The system will sort the answers according to their scores, and it will calculate the precision@N score, where N is the total number of correct answers.

In the following sections, a discussion of each pipeline step is provided, along with the details of the code that represents the annotator.

2.1 Test Elements (SentenceAnnotator)

In this pipeline step, we need to distinguish whether a line of the input file represents either a question or an answer. In this case, we separate each line using a regular expression. Then, we check the first character of the sentence; if it is a 'Q', it means that the text line is a question; otherwise, if it is an 'A', the text line is an answer.

Input Types: N/A

Output Types: Question, Answer

2.2 Tokens (TokenAnnotator)

Here, we iterate over each Question or Answer annotation and splits the text that points each annotation on special characters (whitespace, question mark -?-, colon -.-, comma -,-, admiration sign -!- and simple quote -'-), resulting in substrings. Next, we calculate token indexes by finding the first occurrence index of each substring in the original annotation text. Finally, an offset is added of the index that is equal to the begin index of the original annotation.

Input Types: Question, Answer

Output Types: Token

2.3 N-grams (NGramAnnotator)

For constructing the n-grams, we take the tokens generated in the previous step of the pipeline and we generate lists that contains tokens delimited by newline character. Then, we use each list to generate 1-gram, 2-gram and 3-grams.

Input Types: Token

Output Types: NGram

2.4 Answer Scoring (AnswerScoringAnnotator)

We create two sets, a set of word tokens of the Question and a set for the tokens of an answer. Then, we calculate the Jaccard Index of booth sets and we repeat the process.

Input Types: Token

Output Types: AnswerScore

2.5 Evaluation (EvaluatorAnnotator)

First, we take all AnswerScore annotations and sort the answers by their score in decreasing order. Then, we select the top N answers and calculate the percentage of those answers that are correct. Finally we print the percentage as the precision@N measurement.

Input Types: AnswerScore, Answer

Output Types: N/A

3. Conclusions

In this homework, a processing pipeline is implemented using annotators. All annotators were implemented as Java classes and their respective XML analysis engine descriptor file. Also, an aggregate analysis engine was composed with the annotators described before. Finally the whole pipeline was tested using the sample data provided in the Maven archetype.